



Research paper

# Optimization of atmospheric transport models on HPC platforms

Raúl de laCruz<sup>a</sup>, Arnau Folch<sup>a,\*</sup>, Pau Farré<sup>b</sup>, Javier Cabezas<sup>b</sup>, Nacho Navarro<sup>b,c</sup>, José María Cela<sup>a</sup>

<sup>a</sup> CASE Department, Barcelona Supercomputing Center (BSC), Spain

<sup>b</sup> Computer Sciences Department, Barcelona Supercomputing Center (BSC), Spain

<sup>c</sup> Computer Architecture Department, Universitat Politècnica de Catalunya (UPC), Spain

## ARTICLE INFO

### Article history:

Received 23 September 2015

Received in revised form 19 April 2016

Accepted 25 August 2016

Available online xxx

### Keywords:

High performance computing

Model optimization

FALL3D

Volcanic ash forecast

## ABSTRACT

The performance and scalability of atmospheric transport models on high performance computing environments is often far from optimal for multiple reasons including, for example, sequential input and output, synchronous communications, work unbalance, memory access latency or lack of task overlapping. We investigate how different software optimizations and porting to non general-purpose hardware architectures improve code scalability and execution times considering, as an example, the FALL3D volcanic ash transport model. To this purpose, we implement the FALL3D model equations in the WARIS framework, a software designed from scratch to solve in a parallel and efficient way different geoscience problems on a wide variety of architectures. In addition, we consider further improvements in WARIS such as hybrid MPI-OMP parallelization, spatial blocking, auto-tuning and thread affinity. Considering all these aspects together, the FALL3D execution times for a realistic test case running on general-purpose cluster architectures (Intel Sandy Bridge) decrease by a factor between 7 and 40 depending on the grid resolution. Finally, we port the application to Intel Xeon Phi (MIC) and NVIDIA GPUs (CUDA) accelerator-based architectures and compare performance, cost and power consumption on all the architectures. Implications on time-constrained operational model configurations are discussed.

© 2016 Published by Elsevier Ltd.

## 1. Introduction

Atmospheric Transport Models (ATMs) are used to simulate the transport, dispersion and deposition of a wide spectrum of substances, including those of natural origin (e.g. mineral dust, volcanic ash, aerosols, sea salt), biogenic origin (e.g. biomass burning), or anthropogenic origin (e.g. emission of pollutants, radionuclide leak). Model applications are multiple including, for example, short-term dispersion forecasts of hazardous substances, long-term hazard assessments, air quality evaluations or climate studies. In any case, ATMs always involve three different components: (i) a driving Numerical Weather Prediction Model (NWPM) or meteorological re-analysis dataset (at scales from global to regional), (ii) an emission or source model and, (iii) the transport model *sensu stricto*, which solves a transport equation accounting for advection by wind and diffusion, sedimentation and deposition of the considered substances (other aspects such as chemical reactions, particle interactions or phase changes can also be considered by models depending on each specific problem).

The increase in parallel computing capabilities has allowed ATMs to progressively deal with higher spatial resolutions and coupling mechanisms, although it is well recognized that the performance and scalability of most models on High Performance Computing (HPC) environments is often far from optimal. One reason for this drawback is that model implementations and successive updates have been tradi-

tionally done by physicists and atmospheric scientists, often with little interaction with software engineers. As a result, many existing operational and research codes were not designed from scratch to fully exploit HPC capabilities and its updated versions can not be ported to non general-purpose hardware architectures without having to re-write most of the code, thus precluding efficiency. Here we focus on the particular case of volcanic ash and use the FALL3D model (Costa et al., 2006; Folch et al., 2009) as an example to investigate how different software optimizations and porting to non general-purpose hardware architectures can improve code scalability and execution times.

Explosive volcanic eruptions eject large quantities of particulate matter (tephra) at heights from few to tens of km above the volcano vent. Larger tephra particles typically reach the ground close to the source but finer components, known as coarse ash (particle diameters between 2 mm and 64 µm) and fine ash (particle diameter <64 µm), form volcanic ash clouds that are dispersed by winds aloft at distances from regional to continental before settling on the ground. Volcanic ash clouds jeopardize aerial navigation (Casadevall, 1993) and airports (Guffanti et al., 2009). The global increase in air traffic and number of airports near active volcanoes has raised enormously the risk posed by ash clouds and ash fallout, which cause a high social impact and large economic losses, as dramatically demonstrated during the recent eruptions of Eyjafjallajökull (Iceland, April–May 2010), Grímsvötn (Iceland, May 2011), and Cordón Caulle (Chile, June–July 2011) volcanoes. Observations and ash dispersal model forecasts are essential to support civil aviation management during a volcanic crisis (Folch, 2012). To this purpose, the Volcanic Ash Advisory Centers (VAACs)

\* Corresponding author.

Email address: [afolch@bsc.es](mailto:afolch@bsc.es) (A. Folch)

and other institutions worldwide make use of operational model forecasts to issue advisories and inform civil aviation stakeholders and other parties on future ash cloud location and, in some case, on its mass concentration. From the scientific point of view, a remarkable progress is taking place in the aftermath of the 2010 Eyjafjallajökull event to improve the accuracy of ash dispersal models and reduce/quantify its associated uncertainties (Bonadonna et al., 2012). A major source of forecast uncertainty comes from the characterization of source term (volcanic plume), for which some relevant parameters (e.g. ash injection height, mass eruption rate, and ash particle characteristics) are normally poorly constrained during an eruption. For this reason, one of the emerging strategies is the ensemble forecast, where each member of an ensemble consists on a simulation with a given set of values for those parameters defining the source term (Madankan et al., 2012). The result of an ensemble modeling can be either a probabilistic forecast or a deterministic forecast but with an associated uncertainty (e.g. the dispersion of the ensemble members around the ensemble mean).

In an operational context, code optimization is of interest for, at least, two reasons. On one hand, optimizations under HPC environments allow to run higher-resolution (or larger-domain) model configurations for a given constrained computing time and resources. On the other hand, given a model configuration and grid size, the reduction of the execution times is a must to afford ensemble forecast strategies at operational level, i.e. during an emergency. The aims of this paper are to quantify how different code optimizations and porting to emerging hardware architectures speed up the FALL3D model execution times and to analyze whether the resulting improvements make feasible a future transfer of ensemble forecast strategies into operations. To this purpose, we firstly overview the FALL3D model and apply it to a real test-case, the 2011 Cordón Caulle eruption. Secondly, we present the WARIS framework, in which the FALL3D model governing equations have been implemented (resulting in the so-called WARIS-Transport module), and compare performances of both model implementations for different computational domains and number of particle bins on a general-purpose architecture. The next step is to introduce further optimizations in the WARIS-Transport code, including hybrid MPI-OMP parallelization, algorithmic improvements, spatial blocking, auto-tuning, thread affinity and, finally, porting to Intel Xeon Phi (MIC) and NVIDIA GPUs (CUDA) accelerator-based architectures. Finally, we compare execution times and code strong scalabilities and discuss whether the resulting improvements make feasible a future transfer into operations of ensemble forecast strategies in terms of time and cost.

## 2. The FALL3D model

FALL3D (Costa et al., 2006; Folch et al., 2009) is an Eulerian model for the transport and deposition of volcanic tephra that handles an arbitrary number of particle classes (bins), each characterized by a particle diameter, density and shape (sphericity). FALL3D has a wide community of users worldwide, including the Buenos Aires VAAC (Argentina) which has an operational setup to forecast ash cloud dispersal under its area of influence (e.g. Collini et al., 2013). The model solves one Advection–Diffusion–Sedimentation (ADS) equation for each particle bin using a second-order finite differences explicit scheme on a terrain-following structured grid. FALL3D is coupled off-line with several Numerical Weather Prediction Models (NWPM) and re-analysis datasets. Given the source term, the model outputs the evolution in time of mass concentration and particle accumulation on the ground. The Message Passing Interface (MPI) parallelization of the code is done at two levels, one for the particle bins and another for the computational domain. First, the processors available are distributed amongst groups, each working on one or several bins. In absence of

particle interaction (i.e. if ash aggregation during transport is neglected), this first level of parallelization scales almost linearly because the different groups of processors exchange data only for I/O operations (a single master processor distributes data and gathers results during I/O operations). Second, if a bin has more than one processor assigned, i.e. if the number of processors is a multiple of the number of bins, a second level of parallelization is performed for the computational domain, but only across the vertical direction. This implies the exchange (swapping) of a ghost layer at each time integration step amongst processors working with adjacent domain partitions. However, different aspects preclude FALL3D code performance on HPC environments including sequential Input/Output (I/O), synchronous communications, work imbalance, memory access latency or lack of task overlapping. Note that, even if we focus on one particular ATM, these drawbacks typically exist also in many other operational and research codes.

### 2.1. The 2011 Cordón Caulle eruption test-case

The Puyehue–Cordón Caulle volcanic complex (Chile, 40.5°S, 72.2°W, vent height 1420 m a.s.l.) reawakened on 4 June 2011 at 19 h UTC after decades of quiescence. The initial explosive phase of the eruption, spanning for more than two weeks, was characterized by eruption columns oscillating between 7 and 12 km in height (a.s.l.), generating ash clouds that were dispersed over the Andes causing abundant ash fallout across the Argentinean Patagonia (see e.g. Collini et al., 2013; Bonadonna et al., 2015) for a detailed description of the eruption and the resulting fallout deposit respectively). Ash dispersal was operationally forecasted by the Buenos Aires VAAC using the ETA-HYSPLIT and the WRF/ARW-FALL3D modeling systems. Here we simulate the first three days of the eruption, from 4 June at 19 h UTC to 7 June at 12 h UTC. The idea is to have a real-case reference simulation of sufficient duration to compare the execution times of the original FALL3D model with other model implementations (WARIS-Transport) running on different hardware architectures. The comparison is done in the following sections considering different domain resolutions and number of particle bins.

In order to obtain the driving meteorological data we first run the WRF/ARW meteorological model (Michalakes et al., 2005) at 4 km horizontal resolution ( $950 \times 1250$  horizontal points) and 27 vertical layers using the same physical parameterizations than in Collini et al. (2013). As initial and boundary conditions we used the GFS NCEP FNL (Final) Operational Global Analysis data at 0.5° resolution, available 4 times daily. The WRF/ARW run, with a 6 h of model warm-up, provided hourly outputs to drive the FALL3D model off-line. Meteorological data is interpolated to the FALL3D computational domain, which covers  $30^\circ \times 30^\circ$ , spanning in longitude from 76°W to 46°W and in latitude from 23°S to 53°S. For this domain we consider 3 different cases having spatial resolutions of 0.25, 0.1 and 0.05° respectively (see Table 1) in order to analyze the effect of increasing the size of the computational mesh on the computing times. The coarser domain

**Table 1**

Domain resolutions and number of grid nodes for the 3 different cases considered in the Cordón Caulle reference simulation. Horizontal resolutions along a meridian are approximately 25, 10 and 5 km respectively.

Case name	Horizontal resolution (°)	Vertical resolution (m)	$n_x \times n_y \times n_z$	Number of nodes (millions)
Caulle-0.25	0.25	500	$121 \times 121 \times 64$	0.93
Caulle-0.10	0.10	250	$301 \times 301 \times 64$	5.8
Caulle-0.05	0.05	250	$601 \times 601 \times 64$	23.1

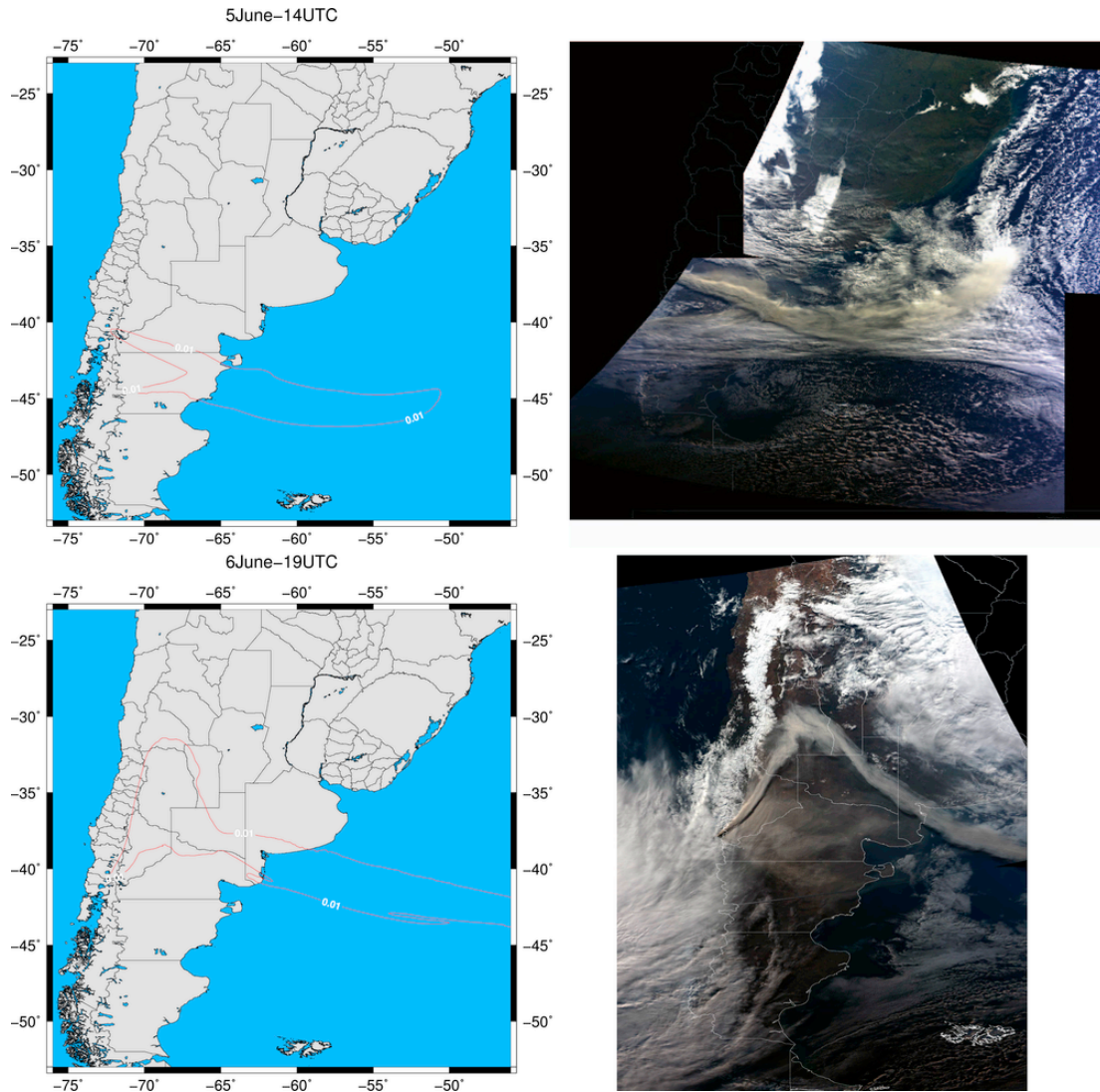
( $\approx 25$  km horizontal resolution along a meridian,  $121 \times 121 \times 64$  nodal points in longitude, latitude and height respectively) can be considered as representative of an operational model set-up. For example, the FALL3D model configuration operational at the Buenos Aires VAAC has a resolution of 24 km and a grid with  $145 \times 170 \times 18$  nodal points. The other two domains are considered in order to study whether a “large” and a “very large” domain could run with computing times compatible with operational time constraints.

As model inputs for this test case, we assume a particle grain size distribution ranging from 1 mm to  $0.7 \mu\text{m}$  and bi-gaussian in  $\Phi$ , with mean- $\Phi$  values of 1 and 3 ( $0.5 \text{ mm}$  and  $125 \mu\text{m}$  respectively) and dispersions of  $0.5\Phi$  and  $1\Phi$ . Note that the  $\Phi$ -scale (e.g. Krumbein, 1934) is defined so that  $d = 2^{-\Phi}$ , where  $d$  is the averaged particle diameter expressed in mm. The simulations consider hourly column heights derived from combining GOES-12 satellite infrared cloud top temperatures and radiosonde thermal profiles launched at Puerto Montt (Chile). These column heights were given to a 1D Buoyant Plume Theory model (Folch et al., 2009, 2016) to obtain the source term (eruption rate and vertical distribution of mass for each particle class) for FAL-

L3D. Although the scope of this paper is not to match simulations and satellite observations, Fig. 1 compares, for illustrative purposes, true color MODIS Terra satellite images with simulations at two different time instants.

### 3. WARIS-Transport

WARIS is Barcelona Supercomputing Center (BSC) in-house multi-purpose framework aimed at solving scientific problems using Finite Difference Methods (FDMs). The framework was designed from scratch to solve in a parallel and efficient way different geoscience problems on a wide variety of architectures. WARIS uses structured meshes to discretize the problem domain, as these are better suited for optimization in accelerator-based architectures. To succeed in such challenge, the WARIS framework was initially designed to be modular in order to ease development cycles, portability, reusability and future extensions of the framework. We have implemented the FALL3D governing equations (Folch et al., 2009) in the WARIS framework, deploying the so-called WARIS-Transport module. The



**Fig. 1.** Comparison between simulated ash column mass (in  $\text{g}/\text{m}^2$ ) and true-color TERRA/AQUA satellite images at two different time instants (5 June at 14 UTC and 6 June at 19 UTC). Mass load has been considered only for particles smaller than  $31 \mu\text{m}$  ( $\Phi = 5$ ) to be consistent with the satellite detection range.

following section details the design of the WARIS framework and its basic internals.

### 3.1. The WARIS framework

A wide variety of hardware architectures is emerging during the last years, from general purpose processors in multi-core and many-core chips (e.g. Intel Xeon) to accelerator-based devices with outstanding performance (e.g. Intel MIC or General-Purpose computing on Graphics Processing Units, GPGPUs). In order to build a framework able to accommodate with ease to any of these architectures, one must ensure an abstraction level of the computational architecture model. To this end, the WARIS framework was designed to include an architecture model with a main entity called *Computational Node* (CN), shown in Fig. 2. This main entity is built using two computational resources: the *host* and the *device* that communicate each other through a *Common Address Space* (CAS) memory. The host resource is responsible for the parallel simulation processes, such as the load balancing (domain decomposition), data communication (exchange of boundary nodes between neighbor domains) and Input/Output operations. The device resource is composed of a set of specialization routines that are used to configure the framework in order to have a functional simulator. The specialization depends on many aspects, such as the physical problem at stack, the hardware platform and the specific numerical method. Within this architecture model, the WARIS framework is executed in the host while the device deals with the specifics of the physical problem being simulated.

Due to performance reasons, WARIS conducts domain decompositions only along the least-stride dimension of the domain ( $Y$  axis in a 3D problem where the dimension-ordered from unit-stride to least-stride dimension is  $Z-X-Y$ ). The reason for this limitation is that it ensures the minimization of gather operations (copies) for sparse data

must be transferred to neighbor domains because data is already arranged in unit-stride layout. The domain slice decomposition is performed in the framework at two different levels: intra-node (node-level) and inter-node (cluster-level). The former level provides decomposition within a CN by means of shared memory, allowing to efficiently exploit platforms with multi-card configuration (e.g. GPUs and MICs). As the memory address space across different CNs is disjoint, the latter level decomposes by means of MPI, adding support for distributed memory implementations of the physical problems. Fig. 3 illustrates this two-level decomposition. To conduct the inter-node communication tasks, the internal nodes adjacent to the boundary nodes are exchanged across neighbors using two communication steps: *front* and *back* (shown in the example of Fig. 3 for sub-domains 0-2/1-0 and 1-2/2-0). This has the advantage of faster data transference but, in contrast, impedes optimal scalability. Finally, in order to run the physical simulation in an efficient and concurrent way, each CN spawns a set of Portable Operating System Interface (POSIX) threads that are independent execution flows in charge of specific tasks. These threads can be classified as:

- *Main thread*: Each CN creates a main thread in charge of orchestrating and commanding the remaining threads spawned within the CN. Its main tasks are reliability and robustness of the infrastructure, as well as allocation and deallocation of resources through control code. The main thread also creates the intra-domains by creating as many *domain threads* as required by the platform specification.
- *Domain threads*: In charge of solving the physical problem by explicitly calling the specialization routines that involve the computational effort. Each domain thread creates an *I/O thread* and as many *Communication threads* as domain neighbors (back and front communication steps).
- *Communication threads*: In charge of performing asynchronous transferences of boundary nodes across neighbor domains. The appropriate parallel paradigm is selected automatically: shared memory for intra-domains, and MPI API for inter-domains (across CNs).
- *I/O threads*: Exclusively in charge of performing I/O operations by means of the I/O library chosen by the user implementation (e.g. POSIX I/O, MPI-IO, HDF5 or NetCDF).

Fig. 4 illustrates a case where the WARIS framework has mapped two intra-domains per each CN. In this example, only one inter-node communication thread is required for domains assigned to sides (Domain 0-0 and Domain 1-1). The hardware architecture model followed by WARIS presents several advantages. First of all, a high level of par-

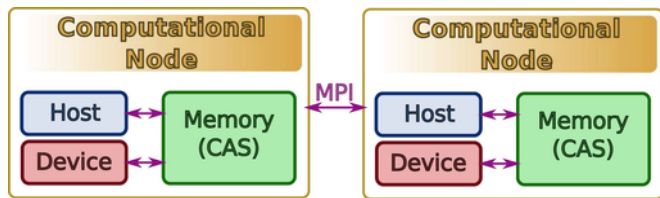


Fig. 2. Architecture model supported by WARIS. A computational node contains a host and the device that communicate through a common address space memory.

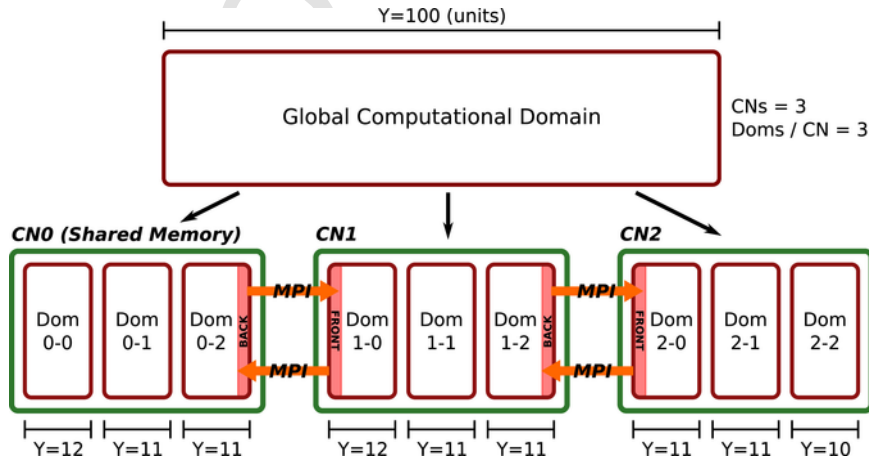


Fig. 3. Levels of domain decomposition in the WARIS framework. The example shows an intra-node level with 3 subdomains within each CN and inter-node level with 3 domains (referenced as 0-\* to 2-\*).



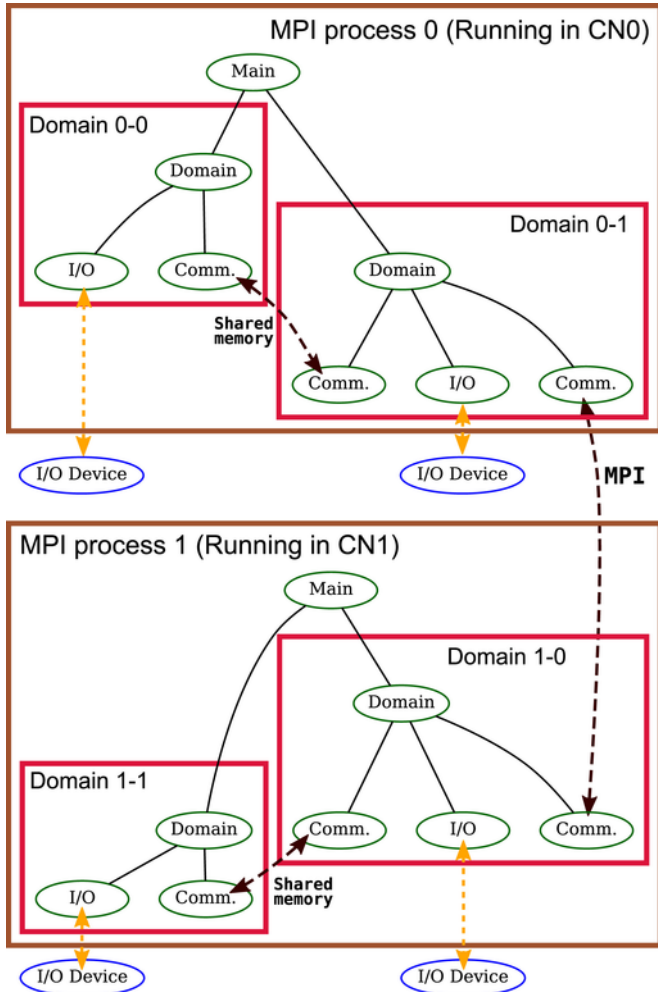


Fig. 4. Threads spawned by the WARIS framework. Example for two CNs, each with two intra-domains. Green and blue nodes represent threads and I/O devices respectively, whereas the brown and red boxes are the MPI processes running on a CN and the intra-node domains within a CN.

allelism that can be achieved by overlapping computation, communication, and Input/Output tasks using independent threads. Second, the flexibility of the abstraction level of the architecture model facilitates the porting to any possible architecture. Finally, unlike FALL3D, the parallelization strategy of WARIS-Transport is only performed through the domain level (i.e. all CNs compute the ADS equation in a slice of the computational domain for all particle bins). This guarantees a better load balancing across CNs disregarding the number of computational nodes and particle bins considered.

### 3.2. FALL3D and WARIS-Transport performance comparison

In order to compare FALL3D and a first naive implementation of WARIS-Transport we run the 2011 Cordon Caulle eruption case study at 3 different spatial resolutions (Table 1).

#### 3.2.1. Testbed platform

As testbed platform, we used the MareNostrum supercomputer facility installed at the BSC. MareNostrum is part of the Partnership for Advanced Computing in Europe (PRACE) research infrastructure as one of the 6 Tier-0 systems available to European scientists. MareNostrum has a peak performance of 1.1 Petaflops, with 48 896 Intel Sandy

Bridge processors in a total of 3056 nodes (16 processors per node). Each MareNostrum node incorporates 2 Intel Sandy Bridge-EP E5-2670 chips (dual socket), a high-end efficient performance processor for servers with 32 GB of memory. Each processor memory hierarchy is composed of a 32 KB L1-I, a 32 KB L1-D, a 256 KB L2 caches on-chip, and a large L3 cache with 20 MB per chip. This Intel architecture also supports Advanced Vector Extensions (AVX), a 256-bit wide Single Instruction Multiple Data (SIMD) instruction set that can compute 4 double-precision floating point instructions per cycle, conferring a peak performance of 166.4 GFlops per chip. All nodes are interconnected through a dual-port high speed network Infiniband FDR10, which uses optical fiber cables and Mellanox 648-port switches for MPI communication. Finally, the supercomputer nodes have access to 2 PetaByte disk storage with General Parallel File System (GPFS), a high-performance shared-disk file system, through 2 additional Giga-bit Ethernet network cards.

#### 3.2.2. Preliminary results

Table 2 summarizes the FALL3D model and the naive (non-optimized) WARIS-Transport execution times for all 3 study cases with 1, 8 and 16 particle bins. All the runs shown in Table 2 used 32 processors (2 entire MareNostrum nodes) with MPI for both intra- and inter-node communication between different domains. Times are also broken down for the four main parts of the simulation: input, output, kernel and others. Input time includes the cost of reading and pre-processing the meteorological data for each hour of simulated time (8 variables of  $n_x \times n_y \times n_z$  and 4 variables of  $n_x \times n_y$  read hourly). Output time considers the execution time of post-processing and writing results every hour of simulated time (13 variables of  $n_x \times n_y \times n_z$  dimension). Note that FALL3D does not have any buffer optimization for I/O operations (i.e. a buffer size adapted to the disc block size) and, consequently, I/O times are much dependent on the disc. Results in Table 2 are given for two different discs of 4 KBy and 16 MBy of block size. Finally, the kernel time refers to the cost of computing explicit kernels such as ADS equation, boundary conditions, ground accumulation (ash fallout), the mass lost at boundaries to perform a mass balance and the correction of the unbalanced mass due to non-null divergence terms.

Looking at results in Table 2 it is observed that WARIS-Transport outperforms FALL3D on all resolutions and number of bins. The speed-ups given by this first preliminary WARIS-Transport implementation range from 7.43 $\times$  in the worst case up to 41.30 $\times$  in the best, depending on the balance between computation and communication (for a given number of processors, the finer the mesh the larger the kernel work load and the lower the speed up). Several reasons explain this higher performance. First, Input and Output (I/O) operations in FALL3D are serial and disc block independent. As a result, only a master process (MPI task 0) performs reading and writing operations and therefore in charge of sending to and receiving from each computational domain the required I/O data. This sequential I/O strategy does not scale because the master process becomes a bottleneck as the number of domains involved in the simulation increases. In opposition, WARIS-Transport uses the parallel implementation of the NetCDF-4 library, which takes advantage of MPI-IO, enabling a much better scaling. Second, the WARIS-Transport module makes use of vectorized explicit kernels (SIMD) that take advantage of the vectorial units of the Intel Sandy Bridge architecture (AVX). This vectorization may speed-up the performance up to 8 $\times$  compared to a scalar implementation when single-precision floating-point operations are carried out (8 FLOPS per AVX operation). In addition, the loops of these explicit kernels have been also rearranged to reduce memory access latency by accessing required data in a proper way. Finally, the multi-threaded ex-

**Table 2**

Computing times (in s) for FALL3D and the naive (non-optimized) version of WARIS-Transport using 32 cores of MareNostrum supercomputer (2 Intel sandy bridge nodes). Times are split between input, output, kernel and others, the later referring to communications (for FALL3D) and to remaining parts of simulation (for WARIS-Transport), *i.e.* initialization, control flow and deallocation of infrastructure. For FALL3D, (1) and (2) differ only in the disc block size (see text for details). The speed-up gives the ratio between FALL3D and WARIS-Transport total execution times.

Domain-bins	CPU time	FALL3D	FALL3D	WARIS-transport	Speed-up
	(s)	(1)	(2)	(no optimized)	
Caulle-0.25-1bin	Input	1914	26	17	–
	Output	54	20	12	–
	Kernel	173	173	22	–
	Others	7	7	1	–
	<b>TOTAL</b>	<b>2148</b>	<b>226</b>	<b>52</b>	<b>41.30(4.34)</b>
Caulle-0.25-8bin	Input	1991	26	27	–
	Output	313	162	15	–
	Kernel	1143	1143	97	–
	Others	11	11	24	–
	<b>TOTAL</b>	<b>3458</b>	<b>1342</b>	<b>163</b>	<b>21.21(8.23)</b>
Caulle-0.25-16bin	Input	1943	26	37	–
	Output	782	299	16	–
	Kernel	2264	2264	205	–
	Others	5	5	26	–
	<b>TOTAL</b>	<b>4994</b>	<b>2594</b>	<b>284</b>	<b>17.58(9.13)</b>
Caulle-0.10-1bin	Input	1125	111	46	–
	Output	410	283	27	–
	Kernel	1814	1814	262	–
	Others	106	106	11	–
	<b>TOTAL</b>	<b>3455</b>	<b>2314</b>	<b>346</b>	<b>9.98(6.68)</b>
Caulle-0.10-8bin	Input	2085	140	71	–
	Output	1279	1313	17	–
	Kernel	9112	9112	1030	–
	Others	49	49	20	–
	<b>TOTAL</b>	<b>12 525</b>	<b>10 614</b>	<b>1138</b>	<b>11.00(9.32)</b>
Caulle-0.10-16bin	Input	3600	98	106	–
	Output	2772	2915	15	–
	Kernel	16492	16492	1684	–
	Others	24	24	18	–
	<b>TOTAL</b>	<b>22 888</b>	<b>19 529</b>	<b>1823</b>	<b>12.55(10.71)</b>
Caulle-0.05-1bin	Input	2335	312	86	–
	Output	524	440	27	–
	Kernel	4782	4782	926	–
	Others	139	139	9	–
	<b>TOTAL</b>	<b>7780</b>	<b>5673</b>	<b>1048</b>	<b>7.43(5.41)</b>
Caulle-0.05-8bin	Input	6325	311	187	–
	Output	4132	3324	46	–
	Kernel	30680	30680	4735	–
	Others	197	197	10	–
	<b>TOTAL</b>	<b>41 334</b>	<b>34 512</b>	<b>4978</b>	<b>8.30(6.93)</b>
Caulle-0.05-16bin	Input	2200	315	325	–
	Output	7256	8313	34	–
	Kernel	68090	68090	7554	–
	Others	224	224	14	–
	<b>TOTAL</b>	<b>77 770</b>	<b>76 942</b>	<b>7927</b>	<b>9.81(9.70)</b>

execution flow of WARIS framework, permits to the Transport module to overlap I/O, MPI communications of boundary nodes and the computation of the explicit kernel. By doing so, a fraction of I/O and MPI exchange of boundary areas can be hidden with the remaining parts of the simulation. Note that, for this reason, I/O times shown in Table 2 for WARIS-Transport refer to these times and cannot be overlapped.

Although the initial naive WARIS-Transport results unveil a considerable speed-up compared to FALL3D, further margin of performance improvement exists for the current HPC scenario. Actually, con-

siderable parallel limitations can be expected in pure MPI applications when strong scaling is performed because of the decreasing ratio of internal nodes to nodes that must be exchanged across neighbor domains (the more MPI decompositions are performed the more dominant communication across domains becomes). Current modern architectures expose high levels of parallelism through the ability of spawning multiple threads that can run simultaneously in the same chip by means of multiple cores. Within the multi- and many-core chips environment, computation and implicit communications can be efficiently conducted at intra-node level by using shared memory paradigms (OpenMP) instead of MPI. To succeed in this challenge, several optimization techniques deploying pseudo-optimal explicit kernels (stencil computations) exploiting effectively not only the high-degree of concurrency in modern HPC architectures but also the shared memory resources (caches) across threads can be applied. Next section considers all these optimization techniques on multi-core chips first and on accelerator-based architectures later on.

#### 4. Further improvements and other architectures

Following the raise of the multi-core paradigm, computing industry has moved from higher clock frequency scaling towards chip multiprocessors to break the limiting wall of performance and energy consumption. In this scenario, conventional cores are replicated and arranged in the same die, sharing memory resources such as cache memories. Nowadays, several mainstream platforms coexist with a variety of architectural features. Programmers have to face multi-core and accelerator-based chips, simultaneous multithreading (SMT), SIMD capabilities, complex memory hierarchies and Non-Uniform Memory Architecture (NUMA) arrangements due to multi-socket implementations. This tremendous parallelism poses optimization challenges. As more of these massively parallel architectures appear in the market, specific parallel strategies have to be explored to achieve suboptimal implementations of the numerical kernels. In this section, we explore a set of architectural optimizations to leverage the performance of the WARIS-Transport module. We present new performance results for three representative and transversal platforms, the previously introduced Intel Sandy Bridge of MareNostrum and the Intel Xeon Phi (MIC) and NVIDIA GPU accelerator-based architectures.

##### 4.1. WARIS optimizations

In order to implement the optimizations and to make them available for future physical specializations (*i.e.* reusable for solving different physical problems), a new support module was implemented and integrated into the WARIS framework. The optimizations included in this module are specially tailored for general purpose processors and multi-core architectures such as the Intel Xeon family (including also the accelerator-based Intel MIC) and include:

###### 4.1.1. OpenMP

The intent is to select an appropriate programming model that maximizes the performance on clusters of nodes with multi-core processors and several sockets. To succeed in this task, the parallelization must be conducted at two different levels: node-level (intra-node) and cluster-level (inter-node). At intra-node level, where the shared memory within the node can be exploited, the OpenMP standard dominates the arena. The OpenMP standard permits a fast prototyping through specific clauses that permit work-sharing constructs of computational areas. Users must annotate their codes with pragmas that specify how computational loops or regions should be parallelized. To conduct this optimization, we first identified the main computational loops of the WARIS-Transport routines. As expected, the most time consuming

routines resulted to be the explicit solver routines and, to a lesser extent, the pre-process and post-process section codes for I/O. Then, we proceeded to annotate with OpenMP pragmas the loops in these routines. The annotation was combined with a thread scheduler which bestowed an appropriate domain decomposition across threads ensuring not only a balanced workload but also an efficient data access to main memory (e.g. streaming access and NUMA aware).

#### 4.1.2. Algorithmic improvements

The second-order finite differences solver of the ADS equation in the WARIS-Transport module can be improved by using novel stencil algorithms (de la Cruz and Araya-Polo, 2014). This novel algorithm takes advantage of a two-phase update of the stencil computation in order to reduce data accesses and improve data reuse on medium and high-order stencils. The implementation of this algorithm in our ADS equation gave us an additional improvement of 10% compared to the classical and vectorized stencil kernel.

#### 4.1.3. Spatial-blocking

In order to leverage the explicit solver performance, we also can incorporate spatial-blocking algorithms into our stencil codes. Space blocking is a widely used technique in computer science that promotes data reuse by traversing data in a specific order. Space blocking is especially useful when the dataset structure to process does not fit into the memory hierarchy. The Rivera blocking (Rivera and Tseng, 2000) proposes a generic blocking scheme for 3D stencil problems. The entire domain is divided and traversed into small blocks of size  $T_I \times T_J \times T_K$  which must fit into the cache memory. A good blocking scheme configuration can be achieved when a  $T_I \times T_J$  2D block is set along the less-stride dimension. Actually, the best configuration is usually given when  $T_I$  is equal to the unit-stride dimension (Kamil et al., 2005). This traversal order reduces cache misses in less-stride dimension accesses, which may increase data locality and overall performance. Note that a search of the best block size parameters ( $T_I \times T_J$ ) must be previously performed for each problem size and architecture configuration.

#### 4.1.4. Auto-tuning

To automatize the spatial-blocking effectiveness, a simple and straightforward auto-tuner was included in our optimization module. This auto-tuner is in charge of finding out possible pseudo-optimal  $T_J$  parameter (local minima) for the spatial-blocking algorithm integrated in our stencil codes. This operation is conducted during the initialization stage of each WARIS-Transport simulation. The cost of conducting this search is marginal (order of milliseconds) compared to the whole simulation time. Selecting an appropriate blocking parameter can save up to 30–40% of the explicit solver for certain simulation cases.

#### 4.1.5. Thread affinity

Due to the multi-threaded and concurrent environment of WARIS-Transport, it is highly recommended to establish a policy of thread affinity within a node. Through this affinity, the spawned OpenMP and WARIS framework threads (*Main*, *Domain*, *Communication* and *I/O* threads) are pinned to specific and different cores, avoiding memory access disruption and interferences across threads. In order to do that, we did not fully populate all hardware cores with OpenMP threads (in charge of intensive computing), leaving some cores exclusively dedicated to WARIS infrastructure management (execution flow, MPI communication and I/O). This thread affinity was achieved by setting properly different environment variables: `OMP_NUM_THREADS` and `KMP_AFFINITY` to control OpenMP threads and their affinity and `EAP_AFFINITY` to pin WARIS framework threads to hardware

cores. On some cases this thread management led to a reduction of 5–15% of the global execution time.

#### 4.2. Intel xeon phi 5110P (MIC)

In 2011 Intel released the first stable Many Integrated Core architecture (MIC), known as Intel Xeon Phi product family, incorporating the research work of the discontinued Larrabee many-core architecture. The 22 nm 5110 P MIC (Knight Corner) processor includes 60 cores with 4-way simultaneous multithreading (SMT) capabilities running at 1.05 GHz. Each core incorporates an in-order execution pipeline and contains a 512-bit wide vector unit (VPU) able to carry out 8 double-precision fused multiply-add operations ( $8 \times 2$  FLOPs) per cycle. This many-core architecture was initially designed for crunching numbers conferring the impressive aggregated peak performance of 1 TFLOPs per chip in SIMD mode. The high SMT/SMP level of MIC (4-way and 60 cores per chip) offers a vast number of parallelism configurations regarding the work-balance distribution among threads (up to 240 threads) and their pinning to cores.

The initial versions of the MIC device were intended to be used as a coprocessor and therefore are required to be attached to a host system. Even though, the MIC platform supports two execution models: native (run from the MIC device) and offload (run from the host device). The former is accessed via the Intel compiler flag `-mmic` which produces an executable targeted specifically for MIC. The latter is based on specific language pragmas that allow the programmer to specify which data must be transferred back and forth from the host to the MIC device.

Our MIC prototype configuration is composed of two Intel Xeon Phi 5110P cards attached to the PCI express bus of a MareNostrum compute node (dual Intel Sandy Bridge-EP E5-2670 chips). Due to the lack of GPFS drivers specifically compiled for MIC architectures, the global disk access is unavoidable configured through a double-mounted system of a NFS server in host which is in turn also mounted to the GPFS system of MareNostrum. The main drawback of this configuration is its poor I/O performance due to the NFS access, which provides a low bandwidth.

The WARIS-Transport specialization for MIC has been deployed using a pure native implementation, where OpenMP and WARIS infrastructure threads are run exclusively in MIC device. The host only provides access to the global disk. As many-core architecture with a cache-based memory hierarchy, the MIC processor can also take advantage of all previous techniques explained in Section 4.1, which were already tested in Intel Sandy Bridge architecture.

#### 4.3. NVIDIA GPUs

In the past, Graphics Processing Units (GPUs) focused exclusively on graphics processing, but since the unification of Vertex shader and Pixel shader units in a single, programmable computing unit, GPUs are able to efficiently execute data-parallel general purpose computations (i.e. CUDA architecture). A GPU typically contains a number of highly multithreaded vector computing cores named streaming multiprocessors (SMs). Each of those SMs contains several vector pipelines that issue instructions that execute on vector functional units (CUDA cores). SMs also contain a large register file, a small L1 data cache, a shared memory, and a specialized cache for textures that can be also used to store read-only data. The execution model is called Single Instruction Multiple Thread (SIMT) where threads are grouped in fixed-length sets of threads (i.e. warps) that execute in lock-step. All threads within a warp (32 in NVIDIA GPUs) execute the same instruction on different data in parallel. Each vector pipeline is able to handle several in-flight warps and has a scheduler that selects the warp to be executed

in each cycle. This helps GPU to hide long latency operations such as cache memory misses.

The CUDA architecture is coupled with a programming model that extends C/C++ to write high performance parallel programs. A CUDA program contains functions that are executed on the CPU, and other functions that are executed in the GPU (called “kernels”). CPU code needs to make sure that data is available in the GPU memory before executing a kernel that accesses it. This is performed via explicit memory transfers that move data through the PCI Express interconnect. Kernel execution and memory transfer operations are asynchronously pushed to the GPU using queues of commands called CUDA streams. Asynchronous operation is key in order to overlap CPU and GPU computations or even GPU memory transfers and GPU kernel execution.

Our GPU machine is an x86 single-node server with an Intel i7-4820 K CPU, 64 GB of DDR3 RAM, 4 NVIDIA Tesla K40 GPUs and a consumer class 1TB HDD. The K40 GPUs implement the GK110 architecture and have the following characteristics: 12 GB GDDR5 288 GB/s, 15 SMs, a 1.5 MB L2 shared cache, and each SM contains four warp schedulers and eight instruction dispatch units (in each cycle, two instructions per warp can be dispatched), 64 K registers, 48 KB shared memory, 16 KB L1 cache, and a 48 KB read-only cache. This configuration allows each K40 GPU to deliver up to 4.29 TFlops in single precision. To compile WARIS-Transport we have used the CUDA Toolkit 7.0 with NVCC compiler and GCC 4.8.

## 5. Results

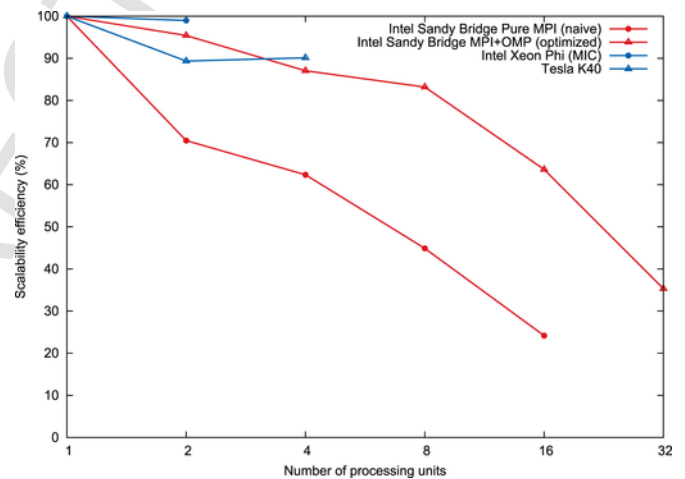
In order to compare performance between different architectures we define a “processing unit” (PU) as the minimum hardware unit available (the minimum hardware unit that is possible to buy). This corresponds, respectively, to 1 general purpose computer node (16 Intel Sandy Bridge processors in our case) or to 1 card for accelerator-based architectures. Table 3 gives the WARIS-Transport total computing times for the Caille-0.05-8bin case on 3 different platforms: general purpose Intel Sandy Bridge (considering the naive pure MPI and the optimized versions), Intel Xeon Phi (MIC) and NVIDIA GPUs (Tesla K40). As observed, the Intel Sandy Bridge optimized version of WARIS-Transport gives additional improvements between 23% and 70% with respect to the pure MPI naive one. The gain increases with the number of PUs because of a much better scalability. The Intel Xeon Phi gives very similar results per processing unit. Finally, the NVIDIA GPUs provide the best performance per PU, improving the optimized general-purpose by a factor of about 2.

**Table 3**

WARIS-Transport simulation times (in s) on 3 different platforms. Values are for the Caille-0.05-8bin case (mesh with 23.1 million nodes, 8 particle classes) considering 3 days of simulation and hourly I/O. Results are given in terms of “processing units”, corresponding to 1 MareNostrum computer node (16 processors) for the testbed platform and to 1 host (node) plus 1 card for Intel Xeon Phi and NVIDIA GPUs (see text for details).

Processing units	Intel Sandy Bridge Pure MPI (naive)	Intel Sandy Bridge MPI+OMP (optimized)	Intel Xeon Phi (MIC)	NVIDIA GPUs (Tesla K40)
1	7015	5368	5633	2917
2	4978	2812	2845	1632
4	2812	1541	—	809
8	1954	806	—	—
16	1815	527	—	—
32	—	475	—	—

Fig. 5 shows the strong scalability efficiency for the same run case on the different platforms. Scalability efficiency is defined as  $100 * t_1 / (n * t_n)$ , where  $t_1$  is the time to complete a work unit with 1 PU and  $t_n$  is the time to complete the same work unit with  $n$  PUs. In the strong scalability analysis the problem size stays fixed while the number of PUs increases, thereby decreasing the computational work per unit and increasing communication. This increases granularity and, in general, degrades parallel performance. Note how the optimized general purpose (Intel Sandy Bridge) version of WARIS-Transport improves very substantially the scalability efficiency, which remains above 80% up to 8 PUs (128 CPUs). The accelerator-based implementations present nearly optimal scalability (above 90%), but our analysis limits to the 4 Tesla K40 cards available. Nevertheless, a significant degradation is observed in the Intel Sandy Bridge scalability with 16 and specially 32 PUs (corresponding to 256 and 512 CPUs respectively). In order to explain the reasons of this lost of performance, Table 4 breaks down the execution time of the Caille-0.05-8bin case with Intel Sandy Bridge platform. Results are categorized in four groups, explicit kernels (*P1*, *P2* and *P3* stages), meteorological input, ash dispersal output and active wait due to synchronous I/O. *P1* (boundary elements), *P2* (internal elements) and *P3* (ash fallout, mass lost, mass balance and mass correction) stages refer to the kernel functions in the WARIS framework structure. *Post-process* and *pre-process* columns consider the required computation for the data arrangement after reading meteorological data and before writing ash dispersal results respectively. Finally, unlike *Input* and *Output* columns, which are asynchronously overlapped (shown in italics), *wait synchronous I/O* includes the time spent on I/O that is serialized (not



**Fig. 5.** Strong scalability efficiency (in %) on different platforms for the Caille-0.05-8bin test case.

**Table 4**

Break-down of the optimized WARIS-transport times (in s) on Intel Sandy Bridge with Caille-0.05-8bin case. Efficiency is shown disregarding the I/O (Comp. eff.) and considering the whole time (Scal. eff.).

Num.	Explicit kernel			Meteo data		Dispersal		Wait	Total	Comp.	Sci.
proc.	P1	P2	P3	Input	Post-	Pre-	Output	sync. I/O	time	eff. (%)	eff. (%)
16	0	2971	2046	816	302	8	65	34	5368	100	100
32	17	1507	1065	438	151	5	79	62	2812	97.0	95.
64	16	778	598	275	76	2	69	67	1541	90.6	87.
128	16	410	190	1	38	1	72	144	806	100	83.
256	16	217	104	1	20	1	113	165	527	93	63.
512	15	126	64	1	11	1	217	255	475	76.4	35.



overlapped with other WARIS-Transport tasks). Due to this, *Input* and *Output* column times are not taken into account to compute the aggregated time (*Total time*). Three main issues arise on these executions. First I/O becomes a bottleneck, specially with a large number of MPI tasks. As the number of processors is increased, the required time for I/O cannot be overlapped with the remaining tasks. The reason is that computing resources are augmented whereas the GPFS I/O servers in MareNostrum remain the same. Second, as the number of domains is increased, each MPI process must read and write smaller chunks of data that entails inefficient gather and scatter operations in the MPI-IO layer. And third, at a certain number of processors, we have experienced serious network contention when MPI routines (point-to-point and collective) and the MPI-IO layer (through NetCDF) are concurrently used by WARIS threads. In order to minimize the MPI contention on executions with more than 128 CPUs, the most critical part of the I/O (71 GB of meteorological data) was previously read in the initialization part of WARIS-Transport. Using this approach, the asynchronous time of *Input* was migrated to *Wait sync. I/O* section, thus reducing the *Input* to only 1 s and increasing considerably the wait time of the simulation due to synchronous I/O. Although this strategy entailed the serialization of I/O and therefore higher *Wait synchronous I/O* values, the whole execution time was paradoxically improved by the fact that the better MPI behavior overcame the cost of serializing the I/O. Finally, Table 5 gives the speed-up factor with respect to the original FALL3D implementation for the Caulle-0.05-8bin case. Certain cases were not feasible to be conducted. For example, the FALL3D and naive WARIS-Transport runs were only executed up to 16 PUs because MPI decomposition limitations ( $n_y$  dimension too small for 512 MPI tasks in 32 PUs case). On the other hand, Intel Xeon Phi (MIC) and Tesla K40 were just limited by the number of available devices. As observed in the Table, speed-ups with respect to FALL3D model are quite remarkable, even with the limiting factor of the I/O performance on large number of processors. In fact, the speed-up is about 55.2 $\times$  for the 16 PUs (256 CPUs).

## 6. Summary and discussion

FALL3D is an Eulerian ATM tailored to simulate transport and deposition of volcanic tephra at both research and operational levels. As occurs for many ATMs, the model implementation in HPC platforms is far from optimal. We have implemented the model equations in the WARIS framework, resulting on a first naive version of WARIS-Transport. Unlike FALL3D, this non-optimal pure MPI implementation handles parallel I/O, asynchronous communications, vectorization and optimal data arrangement to minimize memory access. Using 2 Intel Sandy Bridge nodes (32 CPUs) of the MareNostrum supercomputer, all these aspects decreased the code execution time by a factor varying between 7 and 40 depending on the grid resolution. Further

WARIS-Transport optimizations, including hybrid MPI-OMP parallelization, spatial blocking, auto-tuning and thread affinity lead to much better scalability (efficiency above 65% up to 256 CPUs) and a further gain of 23–70% depending on the number of processors. Finally, we have ported the application to Intel Xeon Phi (MIC) and NVIDIA GPUs (CUDA) architectures. The former gave similar performance results per PU but the later decreased total execution times by a factor of about 2.

From the operational point of view, the importance of code optimization on HPC platforms is clear. On one hand, for a given operational model setup, to dispose of a code at least one order of magnitude faster allows to face ensemble forecast strategies with execution time constraints compatible with operations. On the other, given a limited computing time and computational resources, higher resolution simulations can be considered.

The comparison of performance under different platforms is not straightforward because PUs (general purpose cluster node and accelerator cards) are not directly comparable. For a given problem and execution time other aspects such as hardware cost, power consumption or cost of software development and maintenance should be considered. Table 6 shows a comparison taking into account hardware cost and power consumption to conduct the Caulle-0.05-8bin test. To this extent, we have taken the configuration in number of PUs that entailed similar execution times for each platform. Thus, the configurations selected were 2 PUs for Intel Sandy Bridge (2812 s), 2 PUs for Intel Xeon Phi (2845 s) and 1 PU for NVIDIA Tesla K40 (2917 s). The most efficient configuration is the NVIDIA Tesla K40 with a consumption of only 352 W for the whole execution, followed closely by Intel Xeon Phi with 384 W. Both GPU and MIC implementations require the use of a host. However, while GPU implementation strongly relies on host to perform the remaining tasks of WARIS-Transport (*Main*, *I/O* and *Communication* threads), the MIC implementation only requires the host to provide the power supply (idle consumption of only 36 W for the MIC's host). In addition, the GPU configuration also lead the results in hardware cost terms. With the 40% of the Intel Sandy Bridge or Intel Xeon Phi hardware costs, a deployment of 1 host and 1 NVIDIA Tesla K40 can be acquired providing similar execution times and lesser power consumption. Nevertheless, Table 6 does not detail human effort for porting code from general purpose architectures to GPGPU, which in this case requires more technical knowledge and a different approach than implementing SIMD code with OpenMP pragmas for the other two platforms. Even though, the GPU implementation is a good alternative for VAACs with limited budget in terms of hardware investment and power consumption.

**Table 6**

Watts per execution and cost per platform. The comparison is done for the Caulle-0.05-8bin case with similar execution times between different platforms: 2 PUs of Intel Sandy Bridge, 2 PUs of Intel Xeon Phi and 1 PU of NVIDIA Tesla K40 GPGPU.

	Intel Sandy Bridge	Intel Xeon Phi	NVIDIA Tesla K40
Execution time (s)	2812	2845	2917
Processing units	2 hosts (32 CPUs)	1 host + 2 MICs	1 host + 1 GPU (Tesla K40)
Approximated	$2 \times 5300$	$1 \times 5300 + 2 \times 2500$	$1 \times 1300 + 1 \times 3000$
	=	=	
cost (US\$)	10 600	10 300	4300
Maximum	340 (1 host)	36 (idle host) +	200 (1 host)+
Watts $\times$ hour		225 (1 MIC)	235 (1 GPU)
Watts $\times$ execution	530	384	352

**Table 5**

Speed-up time factor with respect to the FALL3D original implementation for the Caulle-0.05-8bin case considering 3 days of simulation and hourly I/O.

Processing units	Intel Sandy Bridge Pure MPI (naive)	Intel Sandy Bridge MPI+OMP (optimized)	Intel Xeon Phi (MIC)	NVIDIA GPUs (Tesla K40)
1	7.3	9.6	9.1	17.6
2	8.3	14.6	14.6	25.3
4	12.7	23.3	–	44.3
8	16.9	41.0	–	–
16	16.0	55.2	–	–

## Acknowledgments

We thank M.S. Osores from the Argentinean National Scientific and Technical Research Council (CONICET) for providing hourly column heights for the Cordón Caulle eruption simulation and the constructive comments from two anonymous reviewers. This work was supported by NVIDIA through the UPC/BSC GPU Center of Excellence, and the Spanish Ministry of Science and Technology through the TIN2012-34557 project. Finally, we dedicate this work to our colleague and co-author Nacho Navarro, who sadly passed away during the reviewing process.

## References

- Bonadonna, C., Folch, A., Loughlin, S., Puempel, H., 2012. Future developments in modelling and monitoring of volcanic ash clouds: outcomes from the first IAV-CEI-WMO workshop on ash dispersal forecast and civil aviation. *Bull. Volcanol.* 74 (1), 1–10.
- Bonadonna, C., Cioni, R., Pistolesi, M., Elissondo, M., Baumann, V., 2015. Sedimentation of long-lasting wind-affected volcanic plumes: the example of the 2011 Rhyolitic Cordón Caulle eruption, Chile. *Bull. Volcanol.* 77, 2.
- Casadevall, T., 1993. Volcanic hazards and aviation safety: lessons of the past decade. *Flight Saf. Found.-Flight Saf. Dig.*
- Collini, E., Osores, S., Folch, A., Viramonte, J., Villarosa, G., Salmuni, G., 2013. Volcanic ash forecast during the June 2011 cordón caulle eruption. *Nat. Hazards* 66 (2), 389–412.
- Costa, A., Macedonio, G., Folch, A., 2006. A three-dimensional Eulerian model for transport and deposition of volcanic ashes. *Earth Planet. Sci. Lett.* 241 (3–4), 634–647.
- de la Cruz, R., Araya-Polo, M., 2014. Algorithm 942: semi-stencil. *ACM Trans. Math. Softw. (TOMS)* 40 (April (3)), 23:1–23:39. <http://dx.doi.org/10.1145/2591006>.
- Folch, A., Costa, A., Macedonio, G., 2009. Fall3d: a computational model for transport and deposition of volcanic ash. *Comput. Geosci.* 35 (6), 1334–1342.
- Folch, A., Costa, A., Macedonio, G., 2016. FPLUME-1.0: an integral volcanic plume model accounting for ash aggregation. *Geosci. Model Dev.* 9, 1–20.
- Folch, A., 2012. A review of tephra transport and dispersal models: evolution, current status, and future perspectives. *J. Volcanol. Geothermal Res.* 235–236, 96–115.
- Guffanti, M., Mayberry, G., Casadevall, T., Wunderman, R., 2009. Volcanic hazards to airports. *Nat. Hazards* 51 (2), 287–302.
- Kamil, S., Husbands, P., Olike, L., Shalf, J., Yelick, K., 2005. Impact of modern memory subsystems on cache optimizations for stencil computations. In: *MSP'05: Proceedings of the 2005 Workshop on Memory system performance*. ACM Press, New York, NY, USA, pp. 36–43.
- Krumbein, W.C., 1934. Size frequency distributions of sediments. *J. Sediment. Res.* 4 (2), 65–67.
- Madankan, R., Singla, P., Patra, A., Bursik, M., Dehn, J., Jones, M., Pavlonis, M., Pitman, B., Singh, T., Webley, P., 2012. Polynomial chaos quadrature-based minimum variance approach for source parameters estimation. In: *Procedia Computer Science, Proceedings of the International Conference on Computational Science, ICCS 2012*, vol. 9, pp. 1129–1138.
- Michalakes, J., Dudhia, J., Gill, D., Henderson, T., Klemp, J., Skamarock, W., Wang, W., 2005. The weather research and forecasting model: software architecture and performance. In: *Zwiefhofer, W., Mozdzyński, G. (Eds.), Proceedings of the Eleventh ECMWF Workshop on the Use of High Performance Computing in Meteorology*. World Scientific.
- Rivera, G., Tseng, C.W., 2000. Tiling optimizations for 3d scientific computations. In: *Proceedings of ACM/IEEE Supercomputing Conference (SC 2000)*, November, p. 32. URL ([citeseer.ist.psu.edu/rivera00tiling.html](http://citeseer.ist.psu.edu/rivera00tiling.html)).